Privately computing influence in regression models

Adam Richardson
LIA, École Polytechnique Fédérale de Lausanne
adam.richardson@epfl.ch

Ljubomir Rokvic
LIA, École Polytechnique Fédérale de Lausanne
ljubomir.rokvic@epfl.ch

Aris Filos-Ratsikas
University of Liverpool
LIA, École Polytechnique Fédérale de Lausanne
Aris.Filos-Ratsikas@liverpool.ac.uk

Boi Faltings
LIA, École Polytechnique Fédérale de Lausanne
boi.faltings@epfl.ch

Abstract

We consider a crowdsourcing data acquisition scenario, such as federated learning, where a Center C collects data from a set of agents A, with the aim of training a model. To preserve data privacy, the data is not provided directly, but only in aggregated and possibly obfuscated form. To protect itself against low-quality data hidden in the aggregate, the Center may want to evaluate the effect that incorporating the data has on its model accuracy. We propose using a notion of *influence* to make this evaluation. We assume that either the Center has an independent test data set or it relies on data held by another set of agents B. For both linear and logistic regression models, we present a novel method for approximating this influence. We show how this protocol can be implemented efficiently and securely using multiparty computation among A, B, and C, so that A, B and C can assess the quality of individual data items held by A and B without acquiring the data itself.

1 Introduction

The success of machine learning depends to a large extent on the availability of high quality data. For many applications, data has to be elicited from independent and sometimes self-interested data providers. A good example is federated learning (Konečnỳ et al. 2016), where a single *center* (e.g. a large company) collects data from a set of *agents* to jointly learn a model.

An important aspect of federated learning is to preserve the privacy of contributed data, and there are several protocols that allow linear and logistic regression models to be computed without revealing the underlying data. When we consider using these methods in open systems, the privacy-preserving process makes it vulnerable to low quality data contributions. Therefore, it is important that the Center be able to assess the quality of this data before incorporating it into the model. A quality score is useful for various purposes, such as rewarding the data provider, filtering out poor data that would degrade the model, and assessing a data provider's reputation.

Figure 1 shows the setting we assume. Agents A and B each possess private data that is relevant to a model M held by the Center C. To assess the quality of A's data, C computes an estimate of how much incorporating A's data will improve the accuracy of M when evaluated on the data held by Agent B. In a real setting, there will typically be multiple agents of type A, and B could also represent multiple agents.

Such a score can be used by A and B to decide which of their data items are useful to others and thus should be contributed to the federated model, and which of them are outliers that might even degrade the common model. Another use would be to reward A according to the quality of the data it is providing.

The challenge we address in this paper is how to obtain an estimate of this accuracy improvement while preserving the participants' privacy. Specifically: A and B would like to keep their data private from each other and from C.

A clear way of measuring the effect of individual points on the accuracy of a model is via the notion of *influence* (Cook and Weisberg 1980). For a given data point, the influence quantifies how much the model's predictions would change if that point were excluded from the training process. This allows us to quantify the effect that a single data point has on the final outcome; we can simply compare the difference in the model's empirical risk when it is trained with and without the point. The empirical risk is defined as simply the mean of the loss function on some test data set.

For many practical applications, computing the exact influence is prohibitively inefficient. A more serious issue is found in the standard federated learning setting: the Center never obtains the data points from provider A, but only aggregate and possibly obfuscated gradients that help to incorporate the data into the model. Computing the influence of individual points through re-training is generally impossible using only the gradients.

For this reason, following Koh and Liang [2017], we compute an *approximation of the exact influence*, based on upweighing the training point by a small quantity. Our proposed approximation extends the idea in (Koh and Liang 2017), which (without any modifications) turns out to be insufficient for our purposes. We focus on the tasks of estimating distributions, linear and logistic regression and show that the employed approximation is very close to the value of the exact influence, while achieving a notable improvement in speed, especially for logistic regression. We show how these computations can be carried out using multiparty computation between A, B and C without revealing either A or B's data.

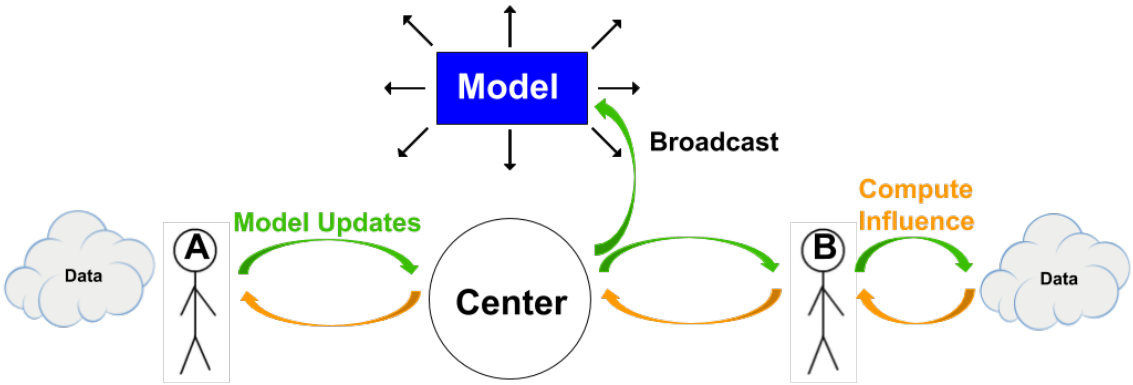

Figure 1: The setting in this paper: Agents A and B hold private data that is relevant to a model M held by the Center C. C wants to know the influence of incorporating A's data on the test data held by B.

Finally, we run experiments on several different real datasets, as well as generated data, and verify our theoretical results.

To our knowledge, we are the first to show that the influence of individual data points can be computed in a secure and privacy-preserving manner. Furthermore, the computation requires no trusted third parties and is efficient enough for large-scale practical use.

1.1 Related Work

Federated Learning Recently, federated learning has become an alternative method to train models that rely on data obtained by many different agents (McMahan et al. 2016). In federated learning, a center coordinates agents who acquire data and provide model updates to a common model maintained by the center. There are many applications in which federated learning can be used, e.g., training models from smart phone data or gathering data from IoT devices and sensors among others. A challenge that the center faces is how to evaluate the usefulness of an agent's model update, while at the same time respecting the agent's privacy, which is a very important issue for federated learning.

An important feature of federated learning is that it generally protects the privacy of the data used for training. There are generally three types of methods used for achieving this:

1. **secure multi-party computation**: the model is learnt by cooperative computation among multiple agents. Each agent only has access to parts of the data, and does not have enough information to infer the rest. There exist many privacy-preserving multi-party computation methods for performing linear and logistic regression using vertically split data, generally based on the technique in (Du, Han, and Chen 2004). The advantage of multi-party computation is that there is usually little computational overhead or restriction of the type of data that can be processed. The disadvantage is that privacy can be broken by collusion among participating agents or by background information about their data.
2. **homomorphic encryption**: each Agent encrypts its data using a homomorphic encryption scheme such as Pallier encryption. The model is computed on the encrypted data; the agents cooperate to decrypt the model but never the data that was used for training. An example of such a method can be found in (Phong et al. 2018). The advantage of such methods is that even with background information, it is not possible to infer the original data. The disadvantage is that it the data has to be mapped to discrete and finite domains, and that there is often large computational overhead to computations with the encrypted data.
3. **differental privacy**: Agents add noise to their data to make it differentially private. The model is computed using this noisy data. An example among many is (Geyer, Klein, and Nabi 2017). The advantage of differential privacy is that it is impossible for anyone to infer the original data, even with background information. The disadvantage is that the noise limits the quality of the model that can be learned.

To avoid the computational overhead of homomorphic encryption, and the inaccuracy associated with differential privacy, we focus on solutions using multiparty computation in this paper. We assume that the Agents and the Center are *honest but curious*, i.e. they don't actively attempt to corrupt the protocol but will try to learn about each other's data.

Influence functions Influence functions are a standard method from robust statistics, which were recently used as a method of explaining the predictions of black box models (Koh and Liang 2017). They have also been used in the context of fast cross-validation in kernel methods and model robustness (Liu, Jiang, and Liao 2014) (Christmann and Steinwart 2004). So far, there has been no work using influence functions to provide quality scores for provided data in a private manner, as we do here.

2 Setting and Foundations

In our setting, there is a *Center* that wants to learn a model parametrized by θ, with a non-negative loss function $L(z, \theta)$ on a sample $z = (\bar{x}, y)$. In this paper, we will assume that

the model is a linear or a logistic regression model. The samples are supplied by a set $\mathcal{A}$ of *Agents*, with agent i providing point $z_i = (\bar{x}_i, y_i)$. Given a set of data $Z = \{z_i\}_{i=1}^n$, the empirical risk, which we use as the loss function to minimize, is $R(Z, \theta) = \frac{1}{n} \sum_i L(z_i, \theta)$.

Influence functions We would like to characterize the quality of a data point by a numerical score. The score should be:

- negative when incorporating the data point will most likely decrease the quality of the model. This allows the center to filter out useless data.
- monotone so that a higher score indicates a higher average accuracy of the resulting model. This allows using the score as an incentive for agents to provide high-quality data.
- the sum of the scores for two sets of data points X and Y that each result in a model of the same accuracy should be the same. This ensures that the score is a reasonable measure for choosing among different data providers.

We now show how to use influence functions to satisfy these criteria. Let $Z_{-j} = \{z_i\}_{i \neq j}$ and let

$$\hat{\theta} = \arg\min_{\theta} R(Z, \theta) \quad \text{and} \quad \hat{\theta}_{-j} = \arg\min_{\theta} R(Z_{-j}, \theta).$$

We will assume that the Center has access to a *test set* $T = \{z_k\}$. The test set may itself reside with another agent B and be subject to privacy constraints, and we will show later how to distribute the computation to satisfy these. Then the *influence* of z_j on the test set is defined as

$$\text{infl}(z_j, T, \theta) = R(T, \hat{\theta}_{-j}) - R(T, \hat{\theta}).$$

We will simply write $\text{infl}(z_j)$, when T and θ are clear from the context. We will use the average influence of the data point on the set of test data as the score of the data point, and note that:

- when the data point decreases the performance of the model, the score is negative.
- the score is proportional to the improvement in the loss function when incorporating the data point.
- we will show later how to scale the score so that the sum of the scores of a set of data approximates closely the decrease in the loss function when incorporating an entire data set.

Therefore, the influence satisfies our desiderata. However, computing it by actually updating the model and comparing the loss functions is not only inefficient, but would require access to the data, and thus violate privacy constraints. In the following, we will show how to compute a very close approximation of the score while respecting the privacy constraints.

3 Approximating Influence

The loss functions which we will use are mean squared error for linear regression and cross-entropy loss for logistic regression. Where y_i is the measured value, and $\hat{y}_i$ is the predicted value for x_i.

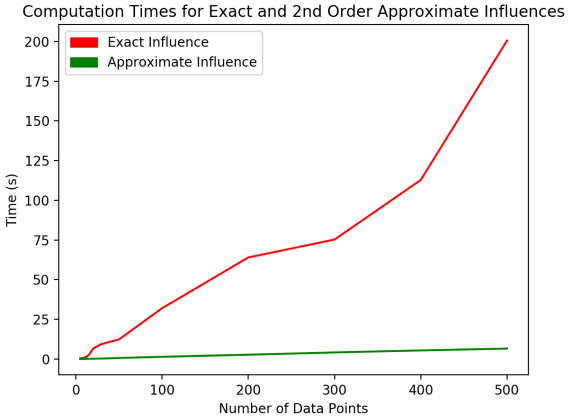

Figure 2: The exact influence is shown to become computationally prohibitive for logistic regression with only a moderate number of data points, while the computation time for the approximate influence increases relatively slowly.

$$L_{MSE} = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

$$L_{CE} = -\frac{1}{n} \sum_{i=1}^{n} (y_i log(\hat{y}_i) + (1 - y_i) log(1 - \hat{y}_i))$$

Trying to practically implement a scoring-based payment mechanism imposes a host of challenges. The first is the computational cost of computing the influence for an agent. Specifically, we must compute $\hat{\theta}_{-j}$, which would involve entirely retraining the model. We present an approximation method based on the method described in (Koh and Liang 2017), which gives the following formula for the influence of z_j on test point z_{test}:

$$\text{infl}(z_{test}, z_j) = \frac{1}{n} \nabla_\theta L(z_{test}, \hat{\theta}) H_\theta^{-1} \nabla_\theta L(z, \hat{\theta})$$

Where H_θ^{-1} is the inverse Hessian computed on all the model's training data. This formula is derived by taking the first term of the Taylor expansion of the empirical risk with respect to θ, which yields an approximation error of $O(1/n^2)$. However, this approximation has the undesirable property that the mean influence is 0, by the definition of $\hat{\theta}$ as the solution to $\sum \nabla_\theta L(z, \hat{\theta}) = 0$. It thus cannot provide an unbiased estimate of the true influence, which is positive in expectation. We obtain a positive average by including the second order term in the Taylor expansion of the empirical risk, and thus obtain an estimate that is close to being unbiased. Let $\partial\theta_j$ be the change in theta due to up-weighting a training point z_j, and let H_i be the Hessian computed only on z_i. We have:

$$\partial\theta_j = \frac{1}{n} H_\theta^{-1} \nabla_\theta L(z_i, \hat{\theta}) + \frac{1}{n^2} H_\theta^{-1} H_i H_\theta^{-1} \nabla_\theta L(z_i, \hat{\theta}) \quad (1)$$

We must also take into account the second order approximation of the change in the loss on a test point when computing the influence. Rather than approximate the change in test loss by $\text{infl}(z_{\text{test}}, z) = (\nabla_\theta L(z_{\text{test}}, \hat{\theta})) \cdot \partial\theta$, we take the second term in the Taylor expansion:

$$\text{infl}(z_{\text{test}}, z) = \left(\nabla_\theta L(z_{\text{test}}, \hat{\theta}) + \frac{1}{2} H_{\theta, z_{\text{test}}} \cdot \partial\theta\right) \cdot \partial\theta \quad (2)$$

Since the Hessian of a function is defined as $\nabla^2 L$ we get the Hessian, in matrix form, for mean squared error and cross-entropy loss in order:

$$H_{MSE} = \frac{2}{n} X^T X$$
$$H_{CE} = \frac{1}{n} X^T D X$$
$$\text{where } D = \text{diag}\left(\sigma(X\theta) \odot (1 - \sigma(X\theta))\right)$$

In general, computing the approximate influence is more efficient than retraining the model and computing the exact influence. This is particularly true in the case of logistic regression. Fig. 2 demonstrates how the computation time for computing the exact influence for logistic regression quickly blows up with the number of data points, as compared to the 2nd Order Approximate influence. The specific implementation of the logistic regressor in our experiments is discussed in Section 6.

4 Secure Influence Computation

Algorithm 1: Influence approximation for linear regression

Data: $\hat{\theta}$, X_i, Y_i, X_{test}, Y_{test}, H_θ
Result: $infl(z_{test}, z_i)$

C: send H_θ^{-1} to A;
C: send $\hat{\theta}$ to A, and B;
A: $\nabla_\theta L(z_i, \hat{\theta}) \leftarrow \frac{2}{n} X_i^T (X_i\hat{\theta} - Y_i)$;
A: $H_{\theta, z_i} \leftarrow \frac{2}{n} X_i^T X_i$;
A: $\partial\theta \leftarrow \frac{H_\theta^{-1} \nabla_\theta L(z_i, \hat{\theta})}{n} + \frac{H_\theta^{-1} H_{\theta, z_i} H_\theta^{-1} \nabla_\theta L(z_i, \hat{\theta})}{n^2}$;
A: send $\partial\theta$ to C;
C: send $\partial\theta$ to B;
B: $H_{\theta, z_{test}} \leftarrow \frac{2}{t} X_{test}^T X_{test}$;
B: $\nabla_\theta L(z_{test}, \hat{\theta}) \leftarrow \frac{2}{n} X_{test}^T (X_{test}\hat{\theta} - Y_{test})$;
B: $infl(z_{test}, z_i) \leftarrow$
$(\nabla_\theta L(z_{test}, \hat{\theta}) + \frac{1}{2} H_{\theta, z_{test}} \cdot \partial\theta) \cdot \partial\theta$;
B: send $infl(z_{test}, z_i)$ to C;
C: send $infl(z_{test}, z_i)$ to A;
A: send H_i to C;
C: update M;
C: update H_θ;

Algorithms Our approach in this situation is to divide the computation so that every party does a part that involves its private data. First, let us clarify *who has what*. Agent A has their own data points z_i which need to be evaluated. Center C has the model parameters θ and the Hessian H_θ computed from past iterations. Party B has its data points z_{test} which will be used to evaluate how beneficial agent A's data points z_i are to the model M. Analyzing the influence approximation formula 2, we find out that:

- To compute $\nabla_\theta L(z_{test}, \hat{\theta})$ the data from parties C, and B is required.
- To compute $H_{\theta, z_{test}}$ the data from party B is required.
- To compute $\partial\theta$ the data from parties A, and C is required.

We will allow party A to compute $\partial\theta$, and H_i. The computed values will then be sent to the center C which will forward $\partial\theta$ to party B. Party B will evaluate the data points z_i using its own test data z_{test}. Following this party B will send the computed influence back to the center C. Then center C will perform the updates on it's model if it sees the data fit.

The data points X_i are an $n \times d$ matrix, while the labels Y_i are an $n \times 1$ matrix. The model parameters $\hat{\theta}$ are a $1 \times d$ matrix, while the H_θ is a $d \times d$ matrix. The test points X_{test} are a $t \times d$ matrix, while the labels Y_{test} are a $t \times 1$ matrix. We also assume that $n >> d$.

Linear regression To be able to compute the influence approximation we have to define $\nabla_\theta L(z_i, \hat{\theta})$ and H_{θ, z_i} for lin-

Algorithm 2: Influence approximation for logistic regression

Data: $\hat{\theta}$, X_i, Y_i, X_{test}, Y_{test}, H_θ
Result: $infl(z_{test}, z_i)$

C: send H_θ^{-1} to A;
C: send $\hat{\theta}$ to A, and B;
A: $\nabla_\theta L(z_i, \hat{\theta}) \leftarrow \frac{1}{n} X_i^T (\sigma(X_i\hat{\theta}) - Y_i)$;
A: $D_i \leftarrow diag(\sigma(X_i\hat{\theta}) \odot (1 - \sigma(X_i\hat{\theta}))$
A: $H_{\theta, z_i} \leftarrow \frac{1}{t} X_i^T D_i X_i$;
A: $\partial\theta \leftarrow \frac{H_\theta^{-1} \nabla_\theta L(z_i, \hat{\theta})}{n} + \frac{H_\theta^{-1} H_{\theta, z_i} H_\theta^{-1} \nabla_\theta L(z_i, \hat{\theta})}{n^2}$;
A: send $\partial\theta$ to C;
C: send $\partial\theta$ to B;
B: $D_{test} \leftarrow diag(\sigma(X_{test}\hat{\theta}) \odot (1 - \sigma(X_{test}\hat{\theta}))$
B: $H_{\theta, z_{test}} \leftarrow \frac{1}{t} X_{test}^T D_{test} X_{test}$;
B: $\nabla_\theta L(z_{test}, \hat{\theta}) \leftarrow \frac{1}{n} X_{test}^T (\sigma(X_{test}\theta) - Y_{test})$;
B: $infl(z_{test}, z_i) \leftarrow$
$(\nabla_\theta L(z_{test}, \hat{\theta}) + \frac{1}{2} H_{\theta, z_{test}} \cdot \partial\theta) \cdot \partial\theta$;
B: send $infl(z_{test}, z_i)$ to C;
C: send $infl(z_{test}, z_i)$ to A;
A: send H_i to C;
C: update M;
C: update H_θ;

ear regression. The formulas are as follows:

$$\nabla_\theta L(z_i, \hat{\theta}) = \frac{2}{n} X_i^T (X_i \hat{\theta} - Y_i)$$
$$H_{\theta, z_i} = \frac{2}{n} X_i^T X_i$$

Algorithm 1 shows a secure implementation on how to evaluate data points z_i in a linear regression model.

Logistic regression Similar to linear regression we also need to define $\nabla_\theta L(z_i, \hat{\theta})$ and H_{θ, z_i}, to be able to compute the approximate influence. The formulas are as following:

$$\nabla_\theta L(z_i, \hat{\theta}) = \frac{1}{n} X_i^T (\sigma(X_i \theta) - Y_i)$$
$$H_{\theta, z_i} = \frac{1}{n} X_i^T D_i X_i$$
$$D_i = \text{diag}\left(\sigma(X_i \hat{\theta}) \odot (1 - \sigma(X_i \hat{\theta}))\right)$$

The resulting algorithm is Algorithm 2.

Maintaining the Hessian H_θ The Hessian matrix H_θ used in the influence computation depends on the data used to train the current model. In the federated learning setting, the Center never has access to this data. However, it can construct H_θ incrementally by averaging the Hessian matrices H_i that agents construct when they contribute a batch of training data. Therefore, we require agents to submit not only the updates to the model parameters, but also the Hessian matrix corresponding to the training data they used.

Security properties Now, let us analyze how much data was disclosed from every party. Agent A shared $\partial\theta$ with the center C, which results in d equations. Additionally, the model update gives d equations, and the update to the Hessian H_i provides d^2 equations. In total, the agent supplies $d \cdot (d+2)$ equations involving the n data points contained in its batch. To ensure that this does not allow backward inference of the data, which has $(d+1) \times n$ parameters, the size of the batch must satisfy:

$$n > \frac{d(d+2)}{d+1}$$

In most regression models, the dimensionality of the data is not very large, so this condition is likely to be satisfied.

Regarding the privacy of the data used to train the current model, the center shares H_θ^{-1} with agent A, which results in $d \times d$ equations. Since we assumed that $n > d$ in all rounds of this process and there are at least $n \times d$ variables, there is again insufficient information to infer the training data.

Finally agent B only shares the resulting approximate influence, resulting in only 1 leaked equation, while the number of parameters agent C poses is equal to $(d+1) \times t$ and thus cannot be derived from this single equation.

We note however that if there is background information regarding the data distribution, the information gained through this process could become sufficient to allow inferences that are correct with high probability. This could be countered by first transforming the data into a lower-dimensional embedding that takes this background information into account.

5 Batch Processing

Computing the influence approximations requires computing the inverse of the Hessian matrix, which is a relatively costly operation. Furthermore, generally an agent will have multiple data points to contribute, and we would like to evaluate the influence of all of them. Thus, it is useful to compute the influence of an entire batch of data points in a single process.

However, the influence of each data point individually is likely to overestimate its actual influence when it is incorporated as part of an entire batch of data. The reason is that there may be redundancies among data points that reduce each data point's influence.

This does not present a problem when comparing the influence of data points supplied by the same agent, but would be important when comparing the quality of batches from different agents, or using the quality score to determine a payment. We wish to incorporate a correction factor that would allow for each data point to be scored as if it were part of its own batch of one. A general way to determine the correction factor is to compare the model's empirical risk before and after incorporating the model update with the sum of the influences for the data points in the batch. However, this can only be done *after* receiving the model updates for the entire batch. In some cases, it might be important to know the correction factor beforehand. We now present a theoretical model that allows to determine the correction factor quite closely. We restrict ourselves to the case of a linear regression model, but the analysis can be extended to any model in which the optimal parameters have a closed form solution.

Let us consider two probability distributions Φ_1 and Φ_2, and we assume they describe an input-output relationship such that $\Phi(x, y) = q(x)p(y|x)$, and $q_1(x) = q_2(x)$. This assumption merely asserts that the data we are collecting is drawn from the same domain regardless of the distribution of the output. Concretely, Φ_1 is the distribution of the data used to derive the current model M, and Φ_2 is the distribution of data held by Agent A.

Distributions Φ_1 and Φ_2 determine, in expectation, models M_1 and M_2 respectively. Let us now define $R_{i,j}$ as the expected risk of model M_i evaluated on distribution Φ_j. Using the standard mean-squared-error loss function, we have that $R_{i,j} = R_{j,j} + \mathbb{E}[(M_i - M_j)^2]$. Now suppose we sample N_1 points from Φ_1 and N_2 points from Φ_2 to form our training set $\{z\}$. Because the linear regression solution is linear with respect to y, and $q(x)$ is fixed, then $\{z\}$ determines in expectation a model $M_c = \frac{N_1 M_1 + N_2 M_2}{N_1 + N_2}$. Then when we evaluate the model, we are only concerned with the error of the mixed model M_c evaluated on Φ_2:

$$R_{c,2} = R_{2,2} + \left(\frac{N_1}{N_1 + N_2}\right)^2 \mathbb{E}\left[(M_2 - M_1)^2\right]$$

To simplify, we fix $N_1 = Q$ as the number of points used for initialization, we define $r = \mathbb{E}[(M_2 - M_1)^2]$, and we let N_2 vary as x. Then we have our expected empirical risk in terms of x:

$$R(x) = \frac{Q^2 r}{(Q + x)^2} + R_{2,2}$$

We can approximate the influence of a data point arriving after x data points as the negative of the derivative of the

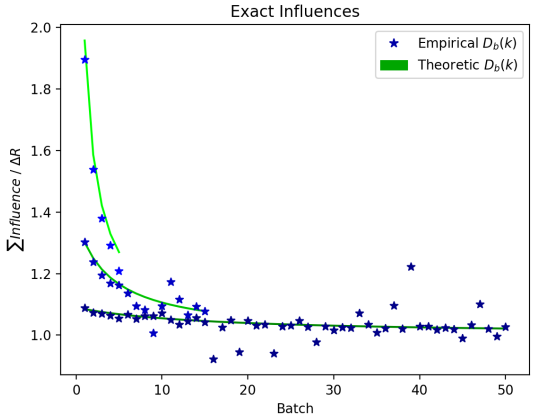

Figure 3: Ratio between Sum of Influences and Change in Loss with respect to batch. Three curves shown representing batch sizes of 30, 100, and 300. 500 points used for initialization.

risk:

$$-\frac{\partial R}{\partial x} = \frac{2Q^2 r}{(Q+x)^3}$$

Now we consider batch size b. We can compute the expected overall change in loss of some arbitrary batch k, with k indexing starting at 1.

$$\Delta R_b(k) = R((k-1)b) - R(kb) = \frac{bQ^2 r(2Q+(k-1)b)^2}{(Q+(k-1)b)^2(Q+kb)^2}$$

Now we consider the sum of influences of points in batch k.

$$S_b(k) = -b\frac{\partial R}{\partial x}\bigg|_{(k-1)b} = \frac{2bQ^2 r}{(Q+(k-1)b)^3},$$

Comparing this to the change in risk, we get the following ratios:

$$D_b(k) = \frac{S_b(k)}{\Delta R_b(k)} = \frac{2(Q+kb)^2}{(Q+(k-1)b)(2Q+(2k-1)b)}$$

By computing these values, the Center can pick an arbitrary batch size and divide the influence scores by this formula such that the expected sum of influences is equal to the overall change in risk, as in the case of batch size 1.

6 Experimental Results

It is important to verify that the approximation used to determine the influence mirrors the actual influence closely enough to be a useful scoring measure. It is also not clear to what extent the model of the distortion caused by batch processing correctly predicts the true correction factor that must be applied.

We therefore conducted experiments with synthetic and real datasets. For training the logistic regressor, we used gradient descent with momentum, with a learning rate of 0.1 and a momentum of 0.9. The convergence criteria was $|\nabla_\theta|_{L1} < \frac{1}{n^6}$ with a cap at 100000 iterations.

Datasets: We start by enumerating the datasets used in our simulations:

- *Linear Generated*: We generate linear regression data as follows: pick an angle θ uniformly in $[-\pi/2, \pi/2]$, and a bias term from $N(0,1)$. Using θ and the bias to determine a linear model, we uniformly sample $x \in [-1,1]$ and determine ground truth y_{gt} values. We then add a noise variable drawn from $N(0,1)$ to produce observations y.
- *Red Wine and White Wine*: UCI datasets with 11 attributes that predict a quality metric. (Cortez et al. 2009)
- *Air Quality*: A UCI dataset with 15 attributes. We removed 6 attributes because they are either non-predictive or they have many missing values. We chose "C6H6(GT)" as the predicted attribute. (De Vito et al. 2008)
- *Logistic Generated*: We generate logistic regression data as follows: pick a slope vector on the unit hyper-sphere and a crossing point inside the half-unit hyper-cube. The negative dot product between these determines the bias term. The bias and slope form a linear model. We uniformly sample $x \in [-1,1]$ and run it through this linear model. We then add a noise variable drawn from $N(0,1)$, and finally threshold at 0 to form the logits.
- *Banknote Authentication (Forgery)*: A UCI dataset with 5 attributes that predict whether or not a banknote is forged. (Dua and Graff 2017)

Approximation Accuracy: We show in Table 1 that on all datasets, our Second Order Approximation formula produces better influence estimates in terms of L1 and L2 error than the First Order Approximation presented in Koh and Liang [2017]. More importantly, this table shows that the mean influence of the First Order Approximation yields a value that is close enough to zero to be interpreted as the result of floating point errors. The one instance this is not the case, Bank Forgery, this is likely due to the logistic regressor not converging adequately.

Batch approximation: We ran simulations to estimate the effect of batch size b on the ratio $D_b(k)$. We ran each simulation with 1500 total training points with a varying batch size. Given a fixed batch size, we ran 10 trials for every dataset and aggregated them to form a more general estimate of $S(k)$, and $\Delta R(k)$. We then took the ratios of these aggregates and compared against our theoretical results for $D_b(k)$ in Fig. 3. We ran this same simulation with different numbers of initial points 20, 100, 200, and 500. We have chosen only to show the case with 500 initial points. The other simulations show the same relationship.

7 Conclusion

There is increasing interest in federated learning to protect the privacy of training data for machine learning. However, this privacy also means that it is no longer possible to ensure that the training data actually improves the model. Outliers and inaccurate data can hurt the performance of the model, and should be excluded. Another issue is that data used for performing this evaluation may also have to remain private.

Data Type	Exact Influence	1st Order Approximation			2nd Order Approximation		
	Mean	Mean	L1 Error	L2 Error	Mean	L1 Error	L2 Error
Linear							
Generated	9.129e-07	5.204e-21	1.792e-06	1.771e-11	9.129e-07	4.883e-12	2.189e-22
Red Wine	2.619e-06	7.744e-15	1.106e-05	8.199e-09	2.619e-06	1.952e-08	3.209e-13
White Wine	2.426e-06	1.665e-14	2.814e-05	1.129e-06	2.481e-06	6.799e-06	1.630e-07
Air Quality	1.377e-05	9.539e-17	3.750e-05	1.652e-06	1.376e-05	5.044e-08	2.598e-11
Logistic							
Generated	3.524e-06	6.553e-19	8.584e-06	4.384e-10	3.846e-06	3.283e-06	8.782e-11
Forgery	2.303e-05	-1.397e-05	8.162e-05	2.226e-05	7.188e-06	4.699e-05	1.263e-06

Table 1: Mean of exact influences, 1st, and 2nd order approximations, along with the mean L1 and L2 errors between the approximations and the exact influences. Linear regression experiments were ran with 1500 training points, while logistic regression experiments were ran with 500. In all cases, 200 points were used for validation.

We have shown how we can use influence functions and multiparty computation to obtain a meaningful score that characterizes the quality of training data. The score can be used to filter bad data, to recognize good and bad data providers, and to pay data providers according to the quality of their data. We have shown how influence can be approximated and processed in batches for efficiency, and developed a theory that allows correcting the difference between the influence and the overall change in loss. We have empirically validated the theoretical results on multiple datasets.

References

Christmann, A., and Steinwart, I. 2004. On robustness properties of convex risk minimization methods for pattern recognition. *Journal of Machine Learning Research* 5(Aug):1007–1034.

Cook, R. D., and Weisberg, S. 1980. Characterizations of an empirical influence function for detecting influential cases in regression. *Technometrics* 22(4):495–508.

Cortez, P.; Cerdeira, A.; Almeida, F.; Matos, T.; and Reis, J. 2009. Modeling wine preferences by data mining from physicochemical properties. *Decision Support Systems* 47(4):547–553.

De Vito, S.; Massera, E.; Piga, M.; Martinotto, L.; and Di Francia, G. 2008. On field calibration of an electronic nose for benzene estimation in an urban pollution monitoring scenario. *Sensors and Actuators B: Chemical* 129(2):750–757.

Du, W.; Han, Y. S.; and Chen, S. 2004. Privacy-preserving multivariate statistical analysis: Linear regression and classification. In *Proceedings of the 2004 SIAM international conference on data mining*, 222–233. SIAM.

Dua, D., and Graff, C. 2017. UCI machine learning repository.

Geyer, R. C.; Klein, T.; and Nabi, M. 2017. Differentially private federated learning: A client level perspective. *arXiv preprint arXiv:1712.07557*.

Koh, P. W., and Liang, P. 2017. Understanding black-box predictions via influence functions. In Precup, D., and Teh, Y. W., eds., *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, 1885–1894. International Convention Centre, Sydney, Australia: PMLR.

Konečnỳ, J.; McMahan, H. B.; Yu, F. X.; Richtárik, P.; Suresh, A. T.; and Bacon, D. 2016. Federated learning: Strategies for improving communication efficiency. *arXiv preprint arXiv:1610.05492*.

Liu, Y.; Jiang, S.; and Liao, S. 2014. Efficient approximation of cross-validation for kernel methods using bouligand influence function. In *International Conference on Machine Learning*, 324–332.

McMahan, H. B.; Moore, E.; Ramage, D.; Hampson, S.; et al. 2016. Communication-efficient learning of deep networks from decentralized data. *arXiv preprint arXiv:1602.05629*.

Phong, L.; Aono, Y.; Hayashi, T.; et al. 2018. Privacy-preserving deep learning via additively homomorphic encryption. *IEEE Transactions on Information Forensics and Security* 13(5):1333–1345.